

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平9-282182

(43) 公開日 平成9年(1997)10月31日

(51) Int.Cl.⁵
G 0 6 F 9/46

識別記号
3 4 0

庁内整理番号

F I
G 0 6 F 9/46

技術表示箇所

3 4 0 B

審査請求 未請求 請求項の数3 OL (全6頁)

(21) 出願番号 特願平8-97997

(22) 出願日 平成8年(1996)4月19日

(71) 出願人 000005108

株式会社日立製作所

東京都千代田区神田駿河台四丁目6番地

(72) 発明者 竹内 理

神奈川県川崎市麻生区王禅寺1099番地 株

式会社日立製作所システム開発研究所内

(72) 発明者 岩崎 正明

神奈川県川崎市麻生区王禅寺1099番地 株

式会社日立製作所システム開発研究所内

(72) 発明者 中原 雅彦

神奈川県川崎市麻生区王禅寺1099番地 株

式会社日立製作所システム開発研究所内

(74) 代理人 弁理士 小川 勝男

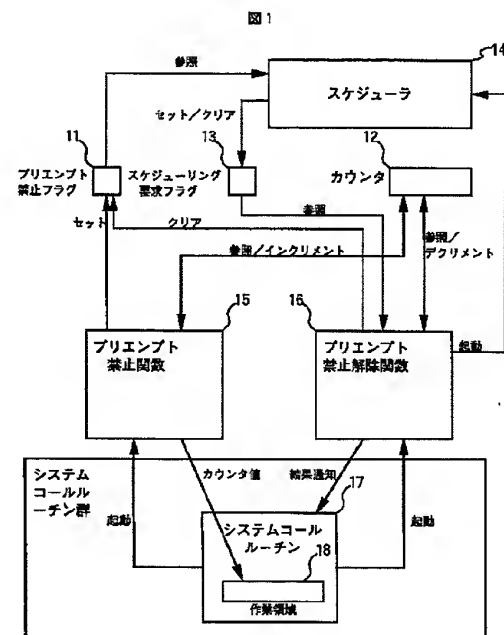
最終頁に続く

(54) 【発明の名称】 計算機システムにおけるプリエンプト制御方法

(57) 【要約】

【課題】 プリエンプトのネスト制御を可能とし、オペレーティング・システム開発の効率を向上させる。

【解決手段】 カウンタを設け、プリエンプト禁止要求があったとき該カウンタ値をインクリメントし、プリエンプト禁止解除の要求があったときに該カウンタ値をデクリメントする。そして、該カウンタ値が0であったならシステムをプリエンプト可能状態にする。一方、カウンタ値が正ならば、プリエンプト禁止状態とする。



【特許請求の範囲】

【請求項1】複数のタスクを並列に実行する計算機システムにおけるプリエンプト制御方法であって、
 実行中タスクの中断禁止を要求する関数を、当該タスクに対応するアプリケーションあるいは当該タスクの実行に係わるオペレーティング・システム内部のルーチンが発行したことに応じて、実行中タスクの中断禁止のネストの数を示すカウンタをインクリメントし、
 実行中タスクの中断禁止の解除を要求する関数を、当該タスクに対応するアプリケーションあるいは当該タスクの実行に係わるオペレーティング・システム内部のルーチンが発行したことに応じて、上記カウンタをデクリメントし、
 前記カウンタの値が正のとき、実行中タスクの中断を禁止し、
 前記カウンタの値が0になったことに応じて、実行中のタスクを中断可能とすることを特徴とする計算機システムにおけるプリエンプト制御方法。

【請求項2】請求項1に記載の計算機システムにおけるプリエンプト制御方法であって、
 タスクのスケジューリング要求が発生したことに応じて、前記カウンタの値を判定し、
 上記カウンタの値が正ならば、スケジューリング動作を実行中タスクが中断可能となるまで遅延させ、
 上記カウンタの値が0ならば、即時にスケジューリング動作を行なうことを特徴とする計算機システムにおけるプリエンプト制御方法。

【請求項3】請求項1あるいは2に記載の計算機システムにおけるプリエンプト制御方法であって、
 実行中タスクの中断禁止を要求する関数のリターン情報として、上記カウンタの更新後の値を返し、
 実行中タスクの中断禁止の解除を要求する関数の引数に、実行中タスクの中断禁止のネストの数を指定し、
 上記引数と上記カウンタの値が一致しなかったことに応じて、該実行中タスクに対応し中断禁止の解除を要求する関数を発行したアプリケーションあるいは当該タスクの実行に係わるオペレーティング・システム内部のルーチンにエラーを通知することを特徴とする計算機システムにおけるプリエンプト制御方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は計算機システムにおけるプリエンプト制御方法に係り、特にマルチタスク環境を提供するオペレーティング・システムに好適な計算機システムにおけるプリエンプト制御方法に関する。

【0002】

【従来の技術】オペレーティング・システムは、その機能をアプリケーション・プログラム（以後、単にアプリケーションと呼ぶ）に提供するために、アプリケーションとの間にシステム・コールと呼ぶインタフェースを設

けている。アプリケーションがシステム・コールを発行すると、オペレーティング・システム内部のルーチン（システムコール・ルーチン）が起動される。ところで、アプリケーションばかりでなくシステムコール・ルーチンを実行中でも、実行中のタスクに割り当てられたCPU時間が使いきられるなどの理由により、タスクの実行が中断され、一定時間間隔にCPUが駆動するルーチンなどから他タスクのスケジューリング要求が発行される可能性がある。このようにタスクの実行が一時的に中断されたとき、実行中のタスクは「プリエンプト」されたと言う。

【0003】アプリケーション又はシステムコール・ルーチンは、危険区域と呼ばれるコード部分を持ち、あるタスクが危険区域内を走行中は、他のいかなるタスクもその危険区域内を走行することは許されない。（ピーターソン／シルバーシャッツ共著、宇津宮孝一／福田晃共訳、「オペレーティングシステムの概念」上巻、培風館、314ページ16行目～319ページ31行目、1987年）。実行中タスクが危険区域を走行中に他タスクのスケジューリング要求が発行された場合、即時に他タスクをスケジューリングしてしまうと、危険区域内を走行中のタスクが存在するにも関わらず、新たにスケジューリングされたタスクがその危険区域を走行する可能性がある。

【0004】従来、危険区域を走行するタスクの上限数を1に制限する手法として、「プリエンプト制御」による手法が知られている。例えばμITRON3.0では、プリエンプト制御を行なうため、以下のインタフェースを提供している。（坂村健監修、「μITRON3.0標準ハンドブック」、パーソナルメディア、74ページ～77ページ、1993年）

(1) dis_dsp：実行中タスクをプリエンプト禁止状態（後述）に遷移させる。

【0005】(2) ena_dsp：実行中タスクをプリエンプト可能状態（後述）に遷移させる。

【0006】アプリケーション又はシステムコール・ルーチンがdis_dsp関数を発行すると、次にena_dsp関数を発行するまで、実行中タスクは「プリエンプト禁止状態」、即ち、他タスクのスケジューリング要求が発行されても、実行中タスクはプリエンプトされない状態に遷移する。dis_dsp関数とena_dsp関数は必ず対になって呼び出される。

【0007】また、アプリケーション又はシステムコール・ルーチンがena_dsp関数を発行すると、実行中タスクは「プリエンプト可能状態」、すなわち、他タスクのスケジューリング要求が発行されれば、即時に実行中のタスクがプリエンプトされる状態に復帰する。また、実行中タスクがプリエンプト禁止状態にある間に発行された他タスクのスケジューリング要求が存在する場合には、このena_dsp関数が発行された時点で他

タスクがスケジューリングされる。

【0008】「プリエンプト制御」とは、実行中タスクのプリエンプト状態（「プリエンプト禁止状態」と「プリエンプト可能状態」の総称）を遷移させることを言う。

【0009】そして、プリエンプト制御により危険区域を走行するタスクの上限数を制限するには、危険区域に入る直前に実行中タスクをプリエンプト禁止状態に遷移させ、かつ、危険区域から抜けた直後にプリエンプト可能状態に復帰させれば良い。

【0010】このようにすると、タスクが危険区域内を走行中はプリエンプト禁止状態に置かれる。よって、あるタスクが危険区域走行中に他タスクがスケジューリングされ、かつそのスケジューリングされたタスクが危険区域内に侵入してくることはありえない。

【0011】

【発明が解決しようとする課題】プリエンプト制御を用いて危険区域を走行するタスクの上限数を制限するため、システムコール・ルーチンにはプリエンプト制御の関数が埋め込まれる。従って、実行中タスクをプリエンプト可能状態に遷移させる関数（`enable`関数）も、システムコール・ルーチンに埋め込まれる。

【0012】いま、実行中タスクがプリエンプト禁止状態にあるときにシステムコール・ルーチンの危険区域内から、他のシステムコール・ルーチン呼び出す場合を考える。呼び出されるシステムコール・ルーチンでは、実行中タスクをプリエンプト可能状態に遷移させる関数を使用されているとする。

【0013】この場合、このシステムコール・ルーチン呼び出し前に、実行中のタスクをプリエンプト禁止状態に遷移させたにもかかわらず、呼び出したシステムコール・ルーチン中で、実行中タスクをプリエンプト可能状態に遷移させる関数が発行されるため、実行中タスクはプリエンプト可能状態に戻ってしまう。すなわち、危険区域内にも関わらず実行中タスクはプリエンプト可能状態になり、プログラムが誤動作する。

【0014】そのため、危険区域内から他のシステムコール・ルーチン呼び出す場合には、呼び出すシステムコール・ルーチンからプリエンプト制御の関数を除いたルーチンを別に用意し、そのルーチン呼び出す必要があり、オペレーティング・システムの開発効率が低下するという問題があった。

【0015】実行中タスクをプリエンプト可能状態に遷移させる関数を使用するライブラリ・ルーチン（複数のシステムコール・ルーチンから使用される共用のルーチン）を危険区域内から呼び出す場合にも同様の問題が発生し、やはりオペレーティング・システムの開発効率が低下する。

【0016】危険区域内からでも、実行中タスクをプリエンプト可能状態に遷移させる関数を使用するシステム

コール・ルーチンやライブラリ・ルーチン（以後、これら二つをまとめてシステムコール・ルーチンと呼ぶことにする）を呼び出し可能とするためには、プリエンプト状態を遷移させる関数の対をネスト可能なプリエンプト制御方法が必要となる。

【0017】本発明の目的は、ネスト可能なプリエンプト制御方法を提供することにある。

【0018】

【課題を解決するための手段】本発明では、複数のタスクを並列に実行する計算機システムにおいて、実行中タスクの実行中断の禁止を要求する関数が、アプリケーション又はシステムコール・ルーチンから発行されたときに、実行中タスクの実行中断の可否を示すカウンタをインクリメントする。また、実行中タスクの実行中断の禁止解除を要求する関数が、アプリケーション又はシステムコール・ルーチンから発行されたときに、上記カウンタをデクリメントする。そして、上記カウンタの値が正であるならば、他タスクへのスケジューリング要求が発生しても、実際のスケジューリング動作を実行中タスクの実行中断の禁止が解除されるまで遅延させる。さらに、上記カウンタの値が0であるならば、他タスクへのスケジューリング要求が発生したら、即時にスケジューリング動作を行なう。

【0019】以上の制御により、プリエンプトのネスト制御が可能となる。

【0020】

【発明の実施の形態】以下、本発明の実施の形態を、図を用いて詳細に説明する。

【0021】図1は、本発明を実現するのに必要なモジュール群、及びデータの概要図である。

【0022】図1において、11はプリエンプト禁止フラグを示す。オペレーティング・システムは、プリエンプト禁止フラグ11のON/OFFによってスケジューラ14の動作を変え、プリエンプト状態を遷移させる。該フラグをONにすると、実行中タスクのプリエンプト状態はプリエンプト禁止状態になり、OFFにするとプリエンプト可能状態になる。

【0023】スケジューラ14のフローチャートを図4に示す。

【0024】スケジューラ14は、まず、ステップ41において、プリエンプト禁止フラグ11を検査する。プリエンプト禁止フラグ11がONであれば、実行中タスクはプリエンプト禁止状態にあることを示す。従って、スケジューラ14は、ステップ42において、スケジューリング要求フラグ13をONにしたら即時にスケジューラを呼び出したルーチン（例えばシステムコール・ルーチン17）にリターンする。

【0025】ステップ41において、プリエンプト禁止フラグ11がOFFであれば、実行中タスクはプリエンプト可能状態にある。従ってスケジューラは、ステップ

43において通常のスケジューリング動作を行ないスケジューラを呼び出したルーチンにリターンする。

【0026】カウンタ12は本発明に従って組み込まれたプリエンプト禁止要求の受理回数を保持するカウンタである。システムコール・ルーチン17から発行されるプリエンプト禁止要求はプリエンプト禁止関数15で処理されるが、その中で、該カウンタ12のインクリメントも行なう。一方、プリエンプト禁止の解除要求はプリエンプト禁止解除関数16において処理され、この中で、該カウンタ12のデクリメントを行なう。これにつ

いては、後ほど詳述する。

【0027】カウンタ12の値が正である場合には、プリエンプト禁止関数15発行回数が、プリエンプト禁止解除関数16の発行回数より上回っていることを示すため、実行中のタスクはプリエンプト禁止状態になければならない。同様に、カウンタ12の値が0である場合には、実行中のタスクはプリエンプト可能状態になければならない。従って、カウンタ12の値が正の場合にはプリエンプト禁止フラグ11がONとなるように、またカウンタの値が0である場合には、プリエンプト禁止フラ

グ11がOFFとなるように、プリエンプト禁止関数15及びプリエンプト禁止解除関数16を実現する。これについても後ほど詳述する。

【0028】システムコール・ルーチン17は、危険区域実現のため、実行中タスクのプリエンプト状態をプリエンプト禁止状態、又はプリエンプト可能状態に移させる必要がある。その場合、以下のインタフェースを用いて、プリエンプト禁止関数15、プリエンプト禁止解除関数16を呼び出す。

【0029】<関数名>

`preempt_disable(*level)`

<引数>

*level:本関数と関数`preempt_enable`で形成される対のネストの深さを受け取るアドレスを指定する。

【0030】<説明>本関数は実行中タスクをプリエンプト禁止状態に移させる。本関数を発行しプリエンプト禁止状態に移移してから、関数`preempt_enable`を発行してプリエンプト可能状態に復帰するまでの区間で、本関数と関数`preempt_enable`を対にして発行しても良い。すなわち、本関数と関数`preempt_enable`の対はネスト可能である。ネストの内側で発行される本関数と関数`preempt_enable`は、プリエンプト禁止状態とプリエンプト可能状態との間の状態遷移は行なわない。*levelで示されるメモリ領域には、このネストの深さが返る。

【0031】<関数名>

`preempt_enable(level)`

<引数>

level:本関数と対となる関数`preempt_disable`が返したネスト・レベルを指定する。

【0032】<説明>本関数は実行中タスクをプリエンプト可能状態に復帰させる。関数`preempt_disable`を発行しプリエンプト禁止状態に移移してから、本関数を発行してプリエンプト可能状態に復帰するまでの区間で、関数`preempt_disable`と本関数を対にして発行しても良い。すなわち、関数`preempt_disable`と本関数の対はネスト可能である。ネストの内側で発行される関数`preempt_disable`と本関数は、プリエンプト禁止状態とプリエンプト可能状態との間の状態遷移は行なわない。levelには、このネストの深さ、すなわち対となる関数`preempt_disable`の発行により得られたlevelの値を指定する。カーネルもこのネストの深さを保持しており、引数で指定されたネストの深さと一致しない場合には、システムコール・ルーチン17にエラーを通知する。

【0033】システムコール・ルーチン17がプリエンプト制御関数を使用する場合には、まずプリエンプト禁止関数15(`preempt_disable`関数)を呼び出した際に、同関数により返される現在のネストの深さを、自関数の作業領域18に格納する。そして、プリエンプト禁止解除関数16(`preempt_enable`関数)を呼び出す際に、その作業領域18に格納されている値を、同関数の引数として指定する。プリエンプト禁止解除関数16では、この引数の値とカウンタ12の値を比較し、両者が一致しない場合にシステムコール・ルーチン17にエラーを通知するため、`preempt_enable`関数と対となるべき`preempt_disable`関数が存在しないというプログラムミス、容易に検出可能になる。

【0034】`preempt_disable`関数の処理手順を図2を用いて説明し、`preempt_enable`関数の処理手順を図3を用いて説明する。

【0035】`preempt_disable`関数では、まず、ステップ21において、カウンタ12の値をインクリメントする。そして、ステップ22において、インクリメント後のカウンタ12の値を、第1引数で与えられたアドレスに複写する。そして、ステップ23において、プリエンプト禁止フラグ11をONにしてリターンする。

【0036】`preempt_enable`関数では、まず、ステップ31において、カウンタ12を参照し、第1引数の値と比較する。その結果、一致していれば、ステップ32においてカウンタ12の値をデクリメントする。一致していなければシステムコール・ルーチン17にエラーを通知する。次に、ステップ33で、デクリメント後のカウンタ12の値を検査する。この時、カウンタ12の値が0でなければ、正常終了する。また、0

であれば、ステップ34においてプリエンプト禁止フラグをOFFにする。さらに、ステップ35において、スケジューリング要求フラグ13を検査し、これがONであれば、ステップ36においてスケジューリング要求フラグ13をOFFにした後、ステップ37でスケジューラを起動し、正常終了する。ステップ35において、スケジューリング要求フラグ13がOFFであれば、ステップ36～37は行なわない。

【0037】上記のような、`preempt_disable`、`preempt_enable`関数を用いれば、両関数の対が複数ネストした場合でも、発行した`preempt_disable`関数と同じ回数だけ`preempt_enable`関数を発行しない限り、実行中タスクはプリエンプト可能状態に復帰しない。従って、危険区域内から（実行中タスクがプリエンプト禁止状態にある場合）でも、実行中タスクをプリエンプト可能状態に遷移させる関数を使用するシステムコール・ルーチン呼び出し可能になる。

【0038】

【発明の効果】以上述べた如く、本発明によれば、プリエンプトの禁止要求とプリエンプトの禁止解除要求の対をネストして発行できる。そして、発行したプリエンプト*

*ト禁止関数と同じ回数だけプリエンプト禁止解除関数を発行しない限り、実行中タスクはプリエンプト可能状態に復帰しない。そのため、実行中タスクがプリエンプト禁止状態にある場合でも、実行中タスクをプリエンプト可能状態に遷移させる関数を使用するシステムコール・ルーチン呼び出し可能になり、オペレーティング・システムの開発効率が向上する。

【0039】また、プリエンプトの禁止要求とプリエンプトの禁止解除要求が正しく対になっていなくても、プリエンプト禁止解除関数のエラーリターンによりそのことが検出可能になっているため、オペレーティング・システムの信頼性を高めることができる。

【図面の簡単な説明】

【図1】本発明を適用したシステムの概要図

【図2】プリエンプト禁止を行う処理のフローチャート

【図3】プリエンプト禁止を解除する処理のフローチャート

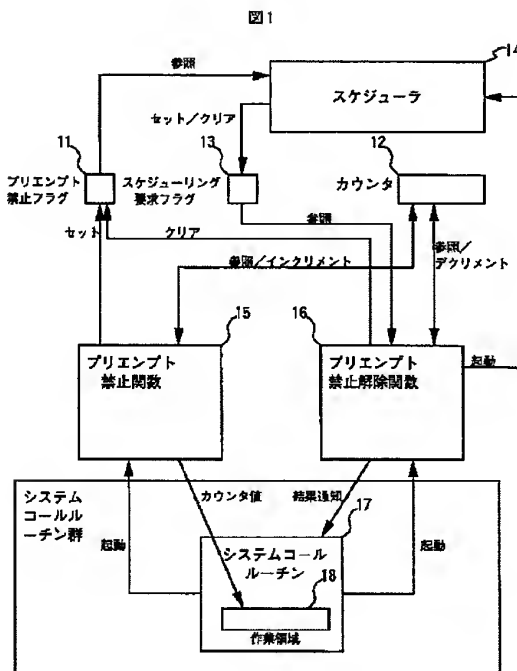
【図4】スケジューラのフローチャート

【符号の説明】

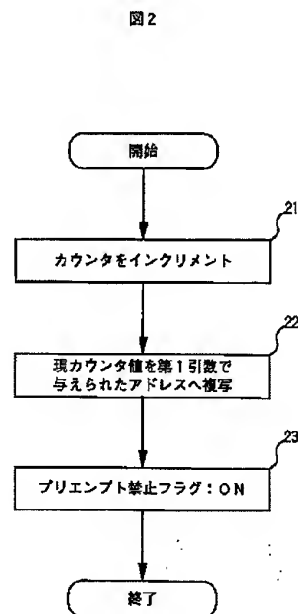
11：プリエンプト禁止フラグ、12：カウンタ、1

3：スケジューリング要求フラグ

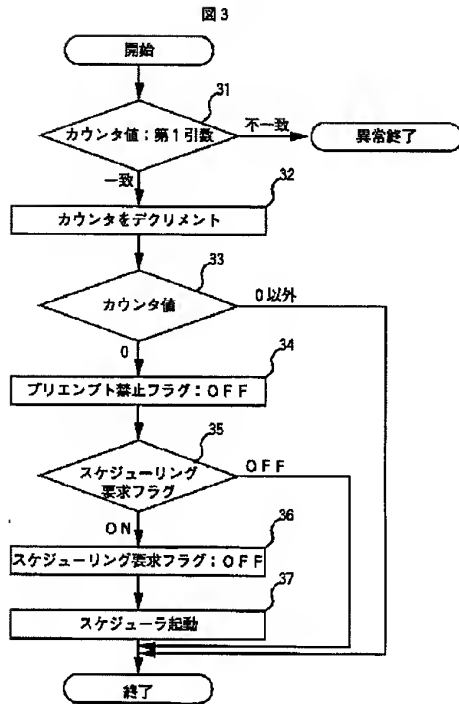
【図1】



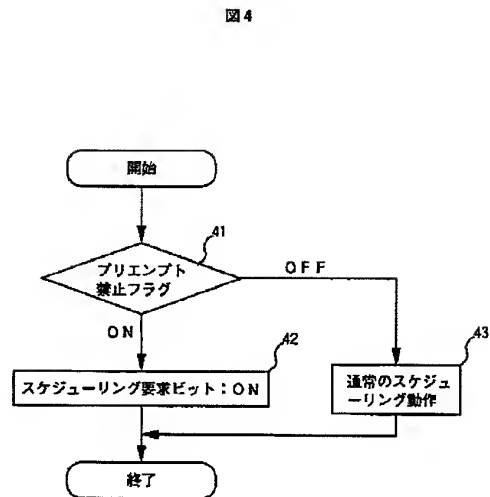
【図2】



【図3】



【図4】



フロントページの続き

(72)発明者 中野 隆裕
 神奈川県川崎市麻生区王禅寺1099番地 株
 式会社日立製作所システム開発研究所内

(72)発明者 芹沢 一
 神奈川県川崎市麻生区王禅寺1099番地 株
 式会社日立製作所システム開発研究所内